A Debugger for Computational Grid Applications

Robert Hood rhood@nas.nasa.gov

Gabriele Jost gjost@nas.nasa.gov

Computer Sciences Corporation
NASA Ames Research Center

http://www.nas.nasa.gov/Tools/p2d2

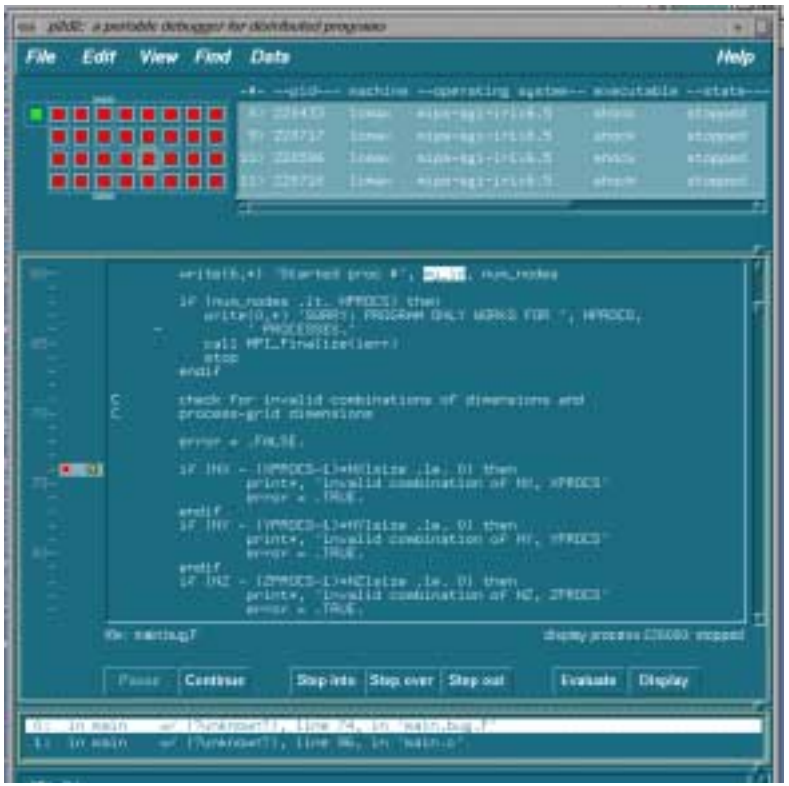
Historical Background

Problem (1994)

wanted a distributed debugger

- with a user interface that scales to “many” processes
- portable across a large variety of machines

Result (1996)



p2d2

(portable parallel/distributed debugger)

- scalable UI
- highly portable
- facilitates further research

1998 Problem:

Needed a debugger for computational grids

Rest of talk

- Architecture for heterogeneity
- Scaling the user interface
- Attaching to grid computations
- Heterogeneity and the user interface
- Status and future work

Accommodating Heterogeneity

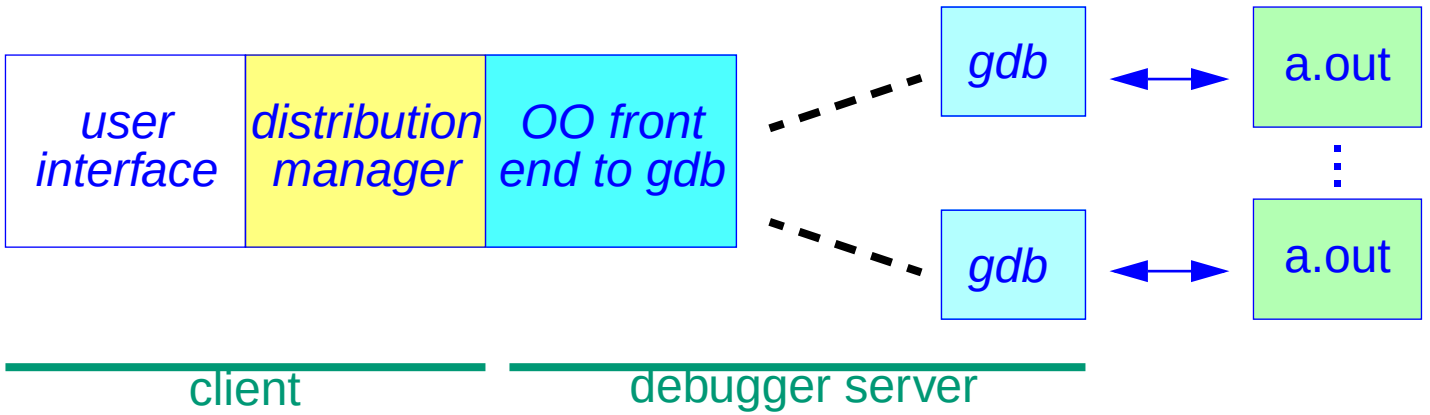
Problem to solve: much of debugger code is not portable

depends on:

- target architecture — trap instruction (breakpoint)
- target OS — access to address space
- compilers — symbol table

Isolate non-portable code in a *Debugger Server*

Initial implementation: use *gdb* as core of server:

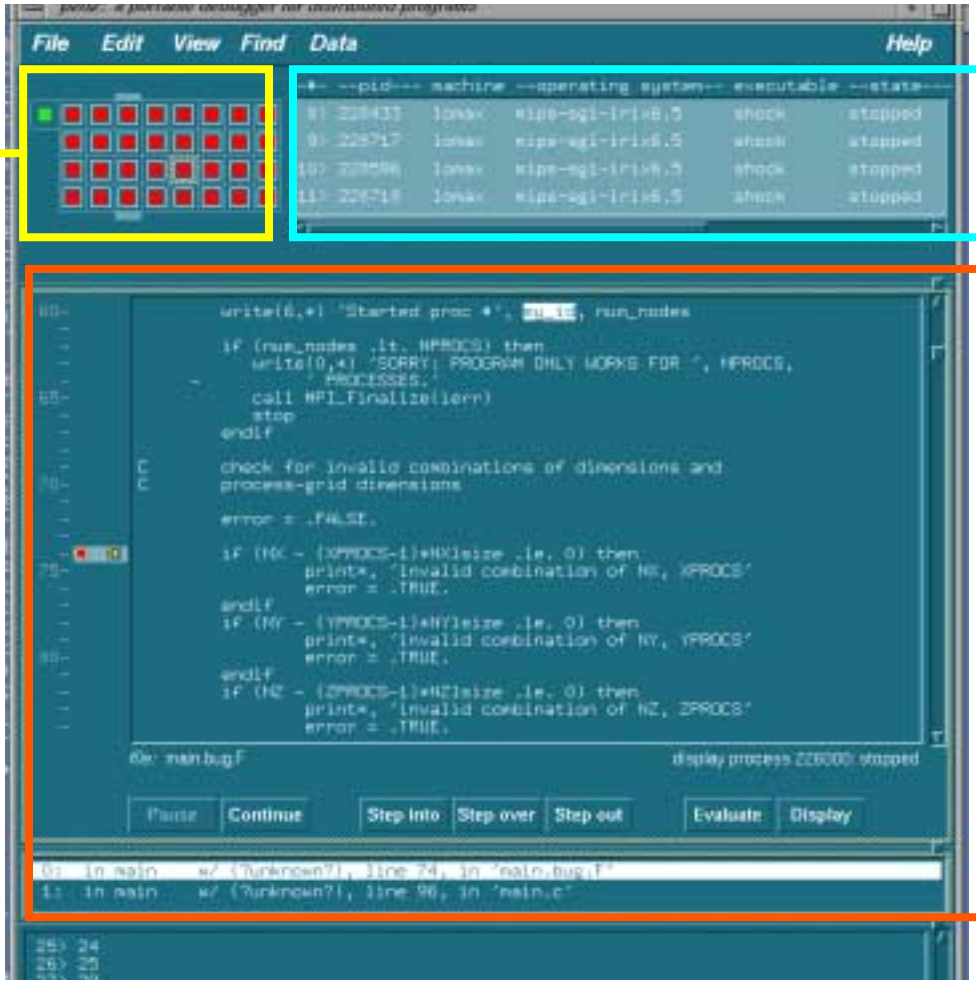


Replication of *gdb*'s permits heterogeneity

Scaling the User Interface

Debuggers have 2 primary functions that need to scale:

- process control - allow collective control
- state examination - provide “zooming”



process grid

focus group

focus process

Attaching to Grid Computations (1)

Debugger needs to gain control of target processes

Case 1: computation initiated by debugger *Run* command

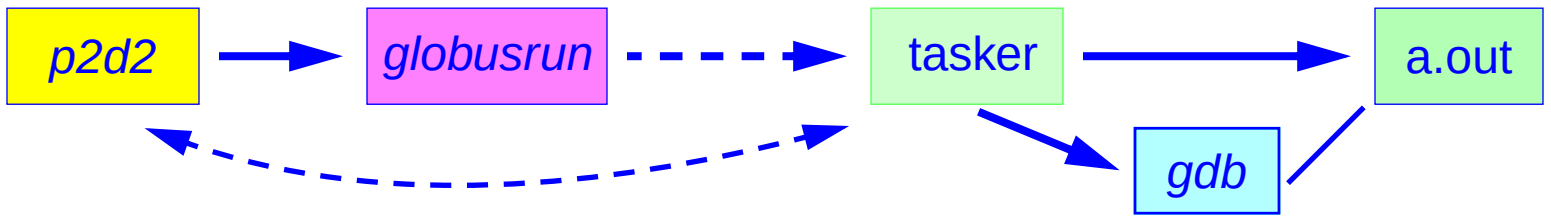
Case 2: existing computation "attached" by debugger

But often some other entity wants to do `fork/exec`

e.g., *mpirun*, *globusrun*, *pvmrun*

Approach for Case 1 (Globus):

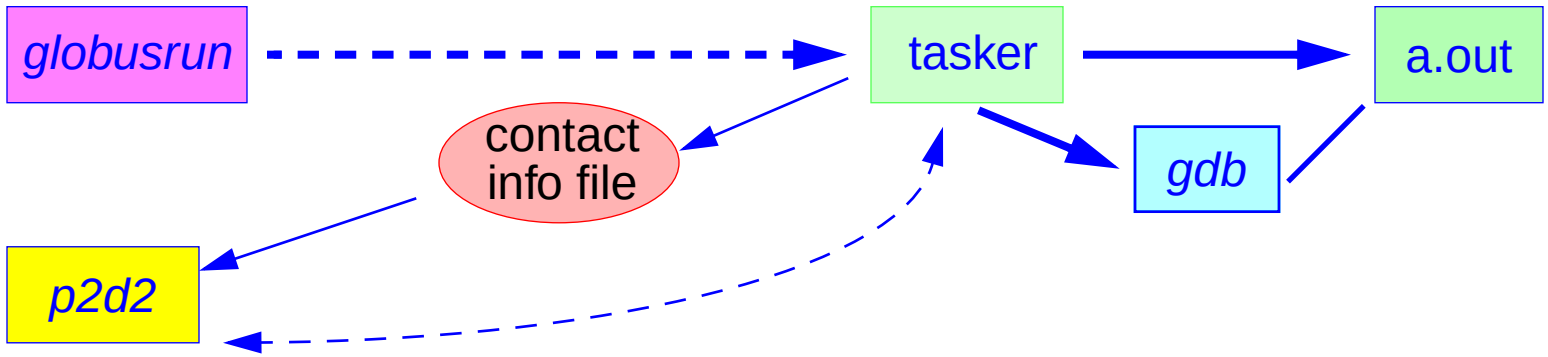
- have *p2d2* create port
- have *p2d2 Run* button start *globusrun*, including RSL: `(paradyn="... port# taskingExecutable")`
- tasker & *p2d2* establish socket
- tasker starts a.out (with special `MPI_Init`); reports pid on socket; a.out sleeps
- *p2d2* asks tasker to start *gdb*; tells *gdb* to attach to a.out



Attaching to Grid Computations (2)

Approach for Case 2 (Globus):

- *Globusrun* creates tasker for each process in job
- each tasker creates port
- port contact information put in file
- *p2d2* reads file & contacts taskers
- *p2d2* asks tasker to start *gdb*; tells *gdb* to attach to a.out



Limitations:

- both cases: must use `paradyn` option in RSL
- case 1: must use special version of `MPI_Init()`

P2d2's Version of `MPI_Init()`

What is does:

- `PMPI_Init()`
- if no tasker present: process 0 writes contact info file:
  (pid, machine, executable) for each process in job
- if initiated from debugger, go into "infinite" sleep loop

When debugger attaches

- have debugger break user process out of loop

Functionality this permits

- run outside of *p2d2*; then start debugger & attach
- startup from within debugger (after setting breakpoints)

Restrictions

- can't stop before user's call to `MPI_Init`
- need to link with special "`.o`" file containing `MPI_Init`
- can't use another library needing to redefine `MPI_Init`

Heterogeneity & the UI—Customizing the Display

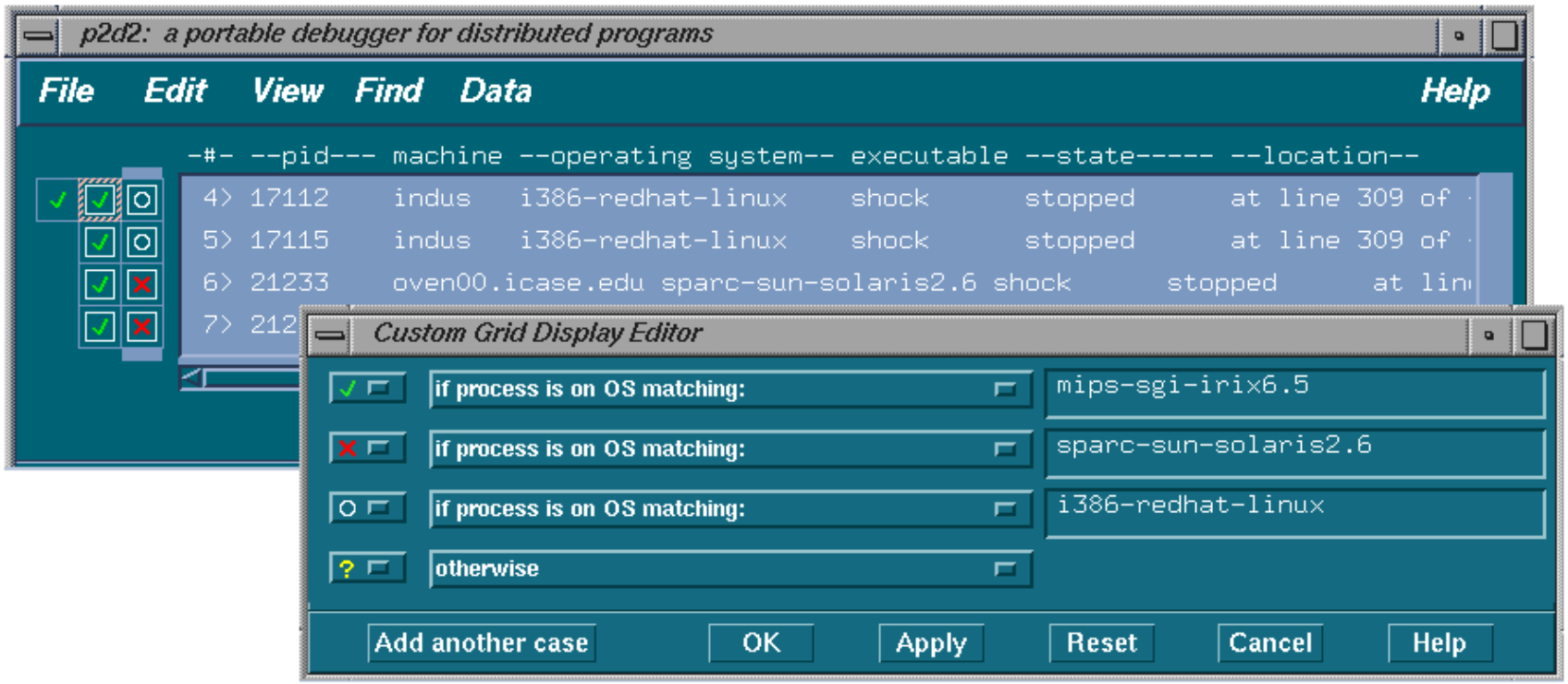
Process grid view can be *programmed*

a list of directives of form: `<icon> if <predicate>`

Samples for `<predicate>`

`running()`, `eval(`*`expr`*`)`, `executingCallTo(`*`fn`*`)`

`systemMatches(`*`string`*`)`



Heterogeneity & the UI—Consistent Data View

Comparing expression values across processes

- *gdb* evaluates to text
- problem: In what context should *gdb* do evaluation?

P2d2 tries to do evaluation in equivalent stack frame

so user is comparing apples to apples

```
#0 in bar_1()        #0 in baz()
#1 in foo()          #1 in bar_2()
#2 in main()         #2 in foo()
                     #3 in main()
```

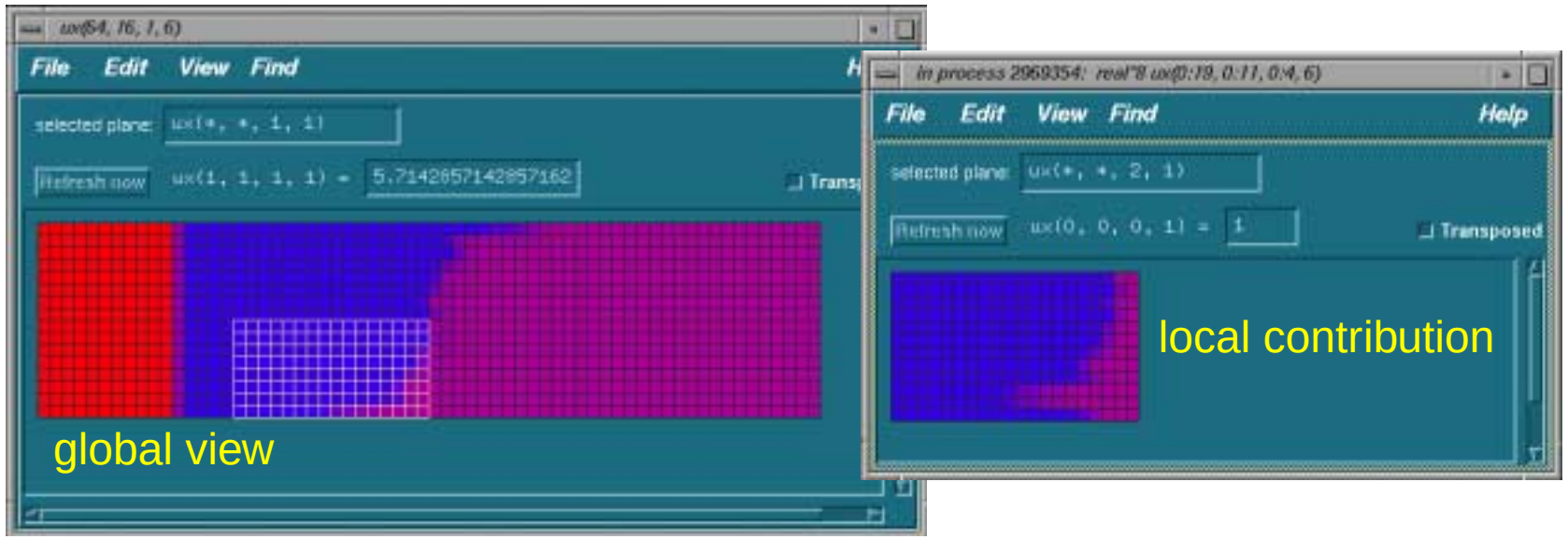
In heterogeneous environment:

- function names don't match, e.g.,

  `foo vs. foo_ vs. foo__`
- so: convert function names to canonical form

Heterogeneity & the UI—Abstract Data View

Distributed array view



Issues:

- Where does distribution info come from?
  - user-provided (via dialog box)
  - derived from parallelization tool database
- *gdb* inconsistencies: e.g. "`whatis a`"
  `real*4 (10,5)` vs `real*4 (5,10)`

Status and Future Work

Status of p2d2 debugging Globus jobs

- SC99 demo: debug a Globus job running on 3 machines
  - SGI Origin in California
  - PC/Linux in Ohio
  - Sparc in Virginia
- controlled a 128-proc Globus job running on 3 Origins
- not yet there:
  - record contact info in MDS
  - security for Globusrun initiated jobs

Distribution status

plan is to distribute under an “Open Source” copyright

Current work

- relative debugging of tool-parallelized programs

Future work

- relative debugging across multiple target platforms